Using Ecological Descriptions to Guide the Construction of Simulation Programs

USING ECOLOGICAL DESCRIPTIONS TO GUIDE THE CONSTRUCTION OF SIMULATION PROGRAMS

D Robertson
A Bundy
M Uschold
B Muetzelfeldt

DAI RESEEARCH PAPER NO. 381

Submitted to ALVEY ANNUAL CONFERENCE/UK IT 88 CONFERENCE,
University College, Swansea, July 1988

USING ECOLOGICAL DESCRIPTIONS TO GUIDE THE CONSTRUCTION OF SIMULATION PROGRAMS

†Dave Robertson, †Alan Bundy, †Mike Uschold, ‡Bob Muetzelfeldt

†Department of Artificial Intelligence, University of Edinburgh, Scotland
‡Department of Forestry and Natural Resources, University of Edinburgh, Scotland

Abstract

We describe a program called EL which helps users with no previous computing experience to construct Prolog programs for running simulations of ecological systems. The modularity provided by the use of Prolog as a target language permits the construction of a database of standard components of simulation program structure (schemata). Each schema enables a program to be built which solves for a single Prolog query (corresponding to an output from the simulation program) and generates a set of subgoals which must be satisfied in order to ensure that the query can be answered from the completed program. This neatly divides the task of program construction into nested sets of goals and subgoals. We could construct simulation programs directly from a database of schemata, using a simple schemata application mechanism but this would put a great burden on users to decide which schemata to apply to a given ecological problem (schemata are quite complex structures). To help reduce the range of schemata which are candidates for representing a particular part of a simulation model we add an extra component to the database – the problem description. The role of the problem description is to represent formally the important features of problems faced by ecological users. EL extracts this information from users with the aid of a knowledge base of ecological and modelling rules. This work has been funded by SERC/Alvey grants GR/C/06226 and GR/D/44294, and (currently) by SERC grant GR/E/00730.

1 Introduction

The EL system is the most recent program constructed as part of the ECO project. The goal of this research is to free ecologists from the need to learn esoteric programming techniques as a prerequisite to investigating the dynamics of ecological systems on a computer. A discussion of these objectives in relation to our current work appears in [1]. Our earlier research emphasised the need for a standard representation of the simulation program which users could manipulate using an intelligent front end package ([2], [3]). This straightforward solution is inadequate in at least two respects. First, it requires users to describe the simulation program directly (albeit in standardised form). Ecologists are frequently unable to construct programs immediately without receiving some guidance, based on a description of their ecological problem. Second, program description formalisms which are manipulated directly by inexperienced users need to be simple to understand. This tends to exclude certain complex program structures from these systems ([4]).

The EL system tackles the first of these problems by explicitly representing important aspects of ecological problems as a precursor to interactive program construction. It also alleviates the second problem, since the use of program schemata permits the manipulation of complex program structures using a simple interface mechanism. Space limitations prohibit more than a thumbnail sketch of EL in this paper. The interested reader is referred to [5] for a thorough description of the mechanisms involved and a complete example of the system in operation. We begin our sketch with a description of how Prolog programs are constructed using the simple schemata application mechanism (section 2) and how the range of applicable schemata may be reduced by reference to problem description information. We then discuss the mechanisms by which a problem description is extracted from the user (Section 3). Finally (in Section 4), we describe a simple method by which the structure of a completed program may be described with respect to the schemata and problem description used in its construction.

2 Constructing Simulation Programs Using Prolog Schemata

Each program schema is a term containing six arguments: its name; a Prolog **goal** for which it solves; a conjunction of **subgoals** which it generates; a set of Prolog **program clauses** which it contributes to the simulation program; a set of procedure calls (**actions**) which must be performed if the schema is selected to contribute to the program ;and a **precondition** which must be satisfied in order for the schema to be used in a given context. Each of these will be described in relation to the schema application mechanism which is used to construct the simulation program.

The diagram in figure 1 illustrates the basic mechanism for schema application, which bears a strong resemblance to the problem solving algorithm used in the MECHO system [6]. Given some Prolog **goal** for which we require a program (represented by goal in the diagram), there will be a number of candidate schemata for supplying this program. In our example, there are only two: schema1

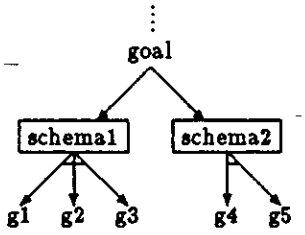

Figure 1: **Schemata application**

and schema2. Each of these schemata will cause a different program to be constructed which can satisfy goal and the user must choose one of them (*i.e.* these are "or" branches of our search tree). Suppose that the user chooses schema1. This generates the **subgoals** g1, g2 and g3 which must be solved, using the same schemata application mechanism, in order to complete the program for goal (*i.e.* these are "and" branches of the search tree). As soon as schema1 is selected any **actions** associated with it are fired. These actions may trigger some interaction with the user (*e.g.* to prompt for the instantiation of particular parameter values) and may also adapt the set of Prolog **program clauses** contributed by the schema to the completed program (*e.g.* there might be an action which constructs clauses representing spatial relations in the program using information obtained from the user). Having executed the action procedures successfully, the program code from schema1 is then incorporated into the developing program and the schemata application mechanism is applied to subgoals g1, g2 and g3. This recursive process terminates when the set of subgoals is empty – a condition which is ensured by deleting any subgoals for which clauses have already been added to the developing program by previous applications of schemata.

Thus far, we have not considered the role which schemata **preconditions** play in the schemata application mechanism. These provide the means by which problem description information reduces the range of candidate schemata. Continuing our example from figure 1, suppose that schema1 and schema2 have preconditions p1 and p2 respectively and that p1 succeeds (with respect to the problem description) while p2 fails. This would exclude schema2 from consideration as a candidate schema for this ecological problem (*i.e.* the "or" branch leading to schema1 is pruned). In this example, only one candidate schema remains so the chosen schema must be schema1. In real sessions there are frequently several candidate schemata with successful preconditions, in which case the user must exercise his/her initiative to choose which of the remaining schemata is most appropriate.

We have demonstrated the value of an explicit problem description in guiding the course of program construction and outlined how this is achieved. However, our efforts are to no avail unless we can provide a mechanism with which users can construct a problem description in the first place. We describe our first working version of this mechanism in section 3.

3 Constructing a Problem Description

We require a formal language for describing ecological problems. As well as possessing the expressive power to cope with standard ecological statements, this language must support mechanisms for guiding users to supply new information which they may have overlooked and prevent them from accidentally supplying contradictory information. We have chosen to use a sorted logic since a number of commonly used ecological statements can be expressed in this formalism ([4]) and guidance can be provided by utilising well understood inference techniques, rather than *ad hoc* methods. The general problem of describing ecological systems is hard because of the wide range of information involved. Fortunately, our task is much simpler because our objective is to find those ecological statements which will help to isolate particular program schemata (and there are a limited number of these). Our discussion will first concentrate on the mechanism with which users may, of their own volition, input ecological statements as sorted logic formulae. We then discuss the guidance mechanisms which are currently provided.

3.1 Locating and Editing Sorted Logic Formulae

EL possesses a set of **template ecological statements** (in the sorted logic) which are expressed over the widest possible range of objects. For example, we provide the formula:

$$\forall X \in object\ \exists A1 \in attribute\ \exists A2 \in attribute \\ varies\_with(A1, X, A2, X) \qquad (1)$$

which is interpreted as "for every object there is some attribute which varies according to an attribute of that object". This statement, as it stands, is too general to be a useful description of a particular ecological problem. Therefore, we provide tools with which users may locate a template formula appropriate to their problem and then restrict the range of objects over which it applies to those which they require.

The sorts referred to in problem description formulae are defined in a sort hierarchy (constrained to be a tree in the current implementation). A browsing subsystem allows users to move through this tree, looking for sorts which apply to their ecological problems. An example display is shown in figure 2, in which the user is positioned at the sort *animal*. The pop–up walking menus in the top panel show the path which the user has followed down through the tree from the most general sort (*universal*) to the current sort (*animal*) and also shows the subsorts of *animal* (*vertebrate* and *invertebrate*). Thus the user is prompted to either explore the subsorts of the current sort or to retrace his/her path back to more general sorts. The sort tree can be extended by the user in two ways. New sorts may be added to the leaves of the tree by moving to the appropriate sort with the browser and typing the

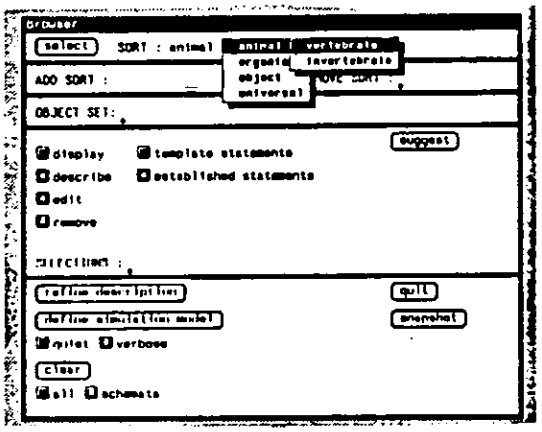

Figure 2: Example of EL Display

name of the new sort at the "ADD SORT" prompt. Alternatively, a sort on a leaf of the tree may be defined as some set of objects by moving to the sort, as before, and typing the set of objects at the "OBJECT SET" prompt.

Having located a sort which interests him/her, the user may add this to a set of selected sorts at the "SELECTIONS" prompt. At any point, the user may instruct EL to find the set of template formulae in which all selected sorts may appear. For example, if the set of template formulae contained only formula 1 and the set of selected names contained the sorts *wolf* and *mass*, then two formulae are extracted:

$$\forall X \in wolf \;\; \exists A1 \in mass \;\; \exists A2 \in attribute \;\; varies\_with(A1, X, A2, X) \quad (2)$$

$$\forall X \in object \;\; \exists A1 \in attribute \;\; \exists A2 \in mass \;\; varies\_with(A1, X, A2, X) \quad (3)$$

Text summaries of each selected formulae are generated using a standard Definite Clause Grammar and presented to the user as items in a pop-up menu. If the user sees an item in this menu which seems relevant to his/her ecological problem then this formula may be passed to an editor for further restriction of substitution sorts and/or addition to the problem description. For example, the user might select the menu item corresponding to formula 2 and restrict the sorts of X, $A1$ and $A2$ to the sort *wolf* and objects *c_mass* (the current mass) and *c_loc* (the current location), respectively, giving the new formula:

$$\forall X \in wolf \;\; varies\_with(c\_mass, X, c\_loc, X) \quad (4)$$

This completes our description of the mechanism with which users may, using their own initiative, add formulae to the problem description. We now discuss the forms of guidance currently provided by EL during this process.

3.2 Guidance Mechanisms

The guidance supplied during problem description is based on two inference mechanisms: deduction and abduction. We consider each separately and then show how they are integrated into the rest of the system. EL contains a collection of rules about ecology and ecological modelling. Some typical examples are:

$$\forall X \in object \;\; \exists A \in attribute \;\; \exists L \in location \;\; varies\_with(A, X, L, X) \rightarrow spatially\_represented(A) \quad (5)$$

$$\forall X \in object \;\; grid\_squares(X) \rightarrow spatially\_represented(X) \quad (6)$$

$$\forall X \in object \;\; irregular\_zones(X) \rightarrow spatially\_represented(X) \quad (7)$$

We have been careful to exclude from our knowledge base rules which can not be interpreted declaratively (*e.g.* condition-action rules) – accepting only those rules for which the consequent will always be true if the antecedent can be established from the problem description. A consequence of this discipline is that any formula deduced from a consistent problem description should, itself, be consistent with that description. For example, if it has been established that the mass attribute *c_mass* of every *wolf* varies according to its current location, *c_loc*, (formula 4), then it must be true (from formula 5) that the model is spatially represented for *wolf*.

Abduction is used to provide suggestions for new formulae which might have caused currently established formulae to be true. Suppose that it has been established that the model is spatially represented for *wolf*. The abduction mechanism allows EL to suggest the possibilities that every *wolf* has a grid square location (using rule 6) or that each *wolf* is assigned to zone locations (from rule 7). It is necessary to prevent overgeneration of suggestions by abduction. Our prime means of doing this is to prohibit abductive generation of formulae which are inconsistent with the current problem description ([7]).

The deduction mechanism is simply incorporated into EL by applying it exhaustively whenever a new formula is added to the problem description. Integration of the abduction mechanism is more complex because the user must choose between the suggestions which it generates. We provide two methods of obtaining suggestions. The first takes place when a user presses the "SUGGEST" button (see figure 2). This generates any abductively derived formulae and offers them to the user as a menu of choices, which the user may then either select for editing or ignore. The second use of abduction occurs when a user has indicated (by pressing the "REFINE DESCRIPTION" button that he/she requires EL to actively take control of describing the ecological problem. EL than finds the set of abductively derived formulae and presents them to the user, as before, but each time the user adds a new formula from the menu of suggestions EL automatically reapplies the abduction mechanism to generate suggestions based on the updated problem description. This process continues until no further abductions can be made.

Once the problem description has been completed to the user's satisfaction, using the methods described above, it is necessary to prime the schemata application mechanism (Section 2) with the Prolog goals corresponding to

required outputs of the simulation program. EL currently performs this task by requiring the user to mark particular problem description formulae as "required outputs". The specially labelled formulae are then converted from sorted to unsorted expressions by producing an instance of an unsorted term for each combination of objects which may be substituted in the arguments of a sorted formula. These unsorted terms are the Prolog goals for which the schemata application mechanism constructs a program.

4 Description of Completed Programs

Having gone to the trouble of accumulating problem description information, it would be wasteful to discard it when the simulation program has been constructed. Such information is useful for subsequent analysis of the program and explanation of its structure. This facility is particularly useful in our case because users are assumed to be unfamiliar with the programming language in which their simulation models are implemented but may still want to know why a particular part of the program was constructed.

We have constructed a subsystem of EL which – given a program which has been constructed by EL; the problem description used during program construction; and a record of the result of any interactions during schemata application – forms a data structure representing the nested application of schemata during the construction of that model and the reasons why each schema was chosen. Users can then move through this data structure (using a browser interface similar to that described in Section 3.1) and examine the actions and preconditions which were satisfied when a particular schema was applied or see the code contributed to the program by a schema. A description of this subsystem appears in [5].

5 Conclusion

In the EL system the task of constructing a simulation program is one of setting up appropriate simulation goals (required outputs of the model) and choosing between alternative program schemata for solving those goals. This has the advantage of shielding from the user much of the computational detail of the program (unless he/she specifically requests to see it). However, it also obliges the system to provide a greater degree of guidance during program construction, since a decision about the correct program structure for a given problem still has to be made, regardless of whether the details of the implementation can be seen by the user. In EL, this guidance is achieved by weeding out schemata which are inappropriate for the user's problem. This, in turn, requires that the aspects of the problem which relate to program construction are represented formally (in our case, in a sorted logic) and that the user is provided with an efficient and friendly way of supplying this information.

EL manages to provide these features using comparatively simple algorithms and standard inference techniques which are independent of the ecology domain (these are detailed in [5]). Nevertheless, the system is still a long way from being a useful product and work continues to extend it in various ways. For example, there is considerable scope for enhancing the problem description mechanisms to provide better initial guidance for inexperienced users. At the other extreme, there is currently no way in which users can modify their problem description once they have begun the program construction phase. It should be possible to detect schemata which *could* have applied, given particular characteristics of the problem description, and initiate a dialogue with the user to attempt to establish these new characteristics.

References

[1] D. Robertson, A. Bundy, M. Uschold, and R. Muetselfeldt. Helping inexperienced users to construct simulation programs: an overview of the ECO project. In *Research and Development in Expert Systems 4*, pages 185–197, British Computer Society Specialist Group on Expert Systems, Cambridge University Press, Brighton, England, 1987. Also available as D.A.I. Research paper 338.

[2] M. Uschold, N. Harding, R. Muetselfeldt, and A. Bundy. An intelligent front end for ecological modelling. In T. O'Shea, editor, *Advances in Artificial Intelligence*, Elsevier Science Publishers, 1986. Also in Proceedings of ECAI-84, and available from Edinburgh University as Research Paper 223.

[3] A. Bundy. Intelligent front ends. In J. Fox, editor, *State of the Art Report on Expert Systems*, pages 15–24, Pergamon Infotech, 1984. also in proceedings of British Computer Society Specialist Group on Expert Systems 1984 and available from Edinburgh as DAI Research Paper 227.

[4] D. Robertson, M. Uschold, A. Bundy, and R. Muetselfeldt. Dialogue in ECO: a system for building ecological simulation models. *submitted to International Journal of Man Machine Studies*, 1988.

[5] D. Robertson, A. Bundy, M. Uschold, and R. Muetselfeldt. *The EcoLogic System.* Working Paper, Department of Artificial Intelligence, University of Edinburgh, 1988.

[6] A. Bundy, L. Byrd, G. Luger, C. Mellish, R. Milne, and M. Palmer. *Solving Mechanics Problems Using Meta-Level Inference.* Research Paper 112, Department of Artificial Intelligence, University of Edinburgh, 1979. In proceedings of IJCAI-6 and Expert Systems in the Micro-electronic age.

[7] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and applications. In J. Siekmann, editor, *Lecture Notes in Computer Science: Proceedings of the 8th International Conference on Automated Deduction*, pages 608–621, Springer-Verlag, 1986.